

PRECONDITIONED ITERATIVE SOLVES IN MODEL REDUCTION OF SECOND ORDER LINEAR DYNAMICAL SYSTEMS

NAVNEET PRATAP SINGH, KAPIL AHUJA, AND HEIKE FASSBENDER

ABSTRACT. Recently a new algorithm for model reduction of second order linear dynamical systems with proportional damping, the Adaptive Iterative Rational Global Arnoldi (AIRGA) algorithm [8], has been proposed. The main computational cost of the AIRGA algorithm is in solving a sequence of linear systems. These linear systems do change only slightly from one iteration step to the next. Here we focus on efficiently solving these systems by iterative methods and the choice of an appropriate preconditioner. We propose the use of relevant iterative algorithm and the Sparse Approximate Inverse (SPAI) preconditioner. A technique to cheaply update the SPAI preconditioner in each iteration step of the model order reduction process is given. Moreover, it is shown that under certain conditions the AIRGA algorithm is stable with respect to the error introduced by iterative methods. Our theory is illustrated by experiments. It is demonstrated that SPAI preconditioned Conjugate Gradient (CG) works well for model reduction of a one dimensional beam model with AIRGA algorithm. Moreover, the computation time of preconditioner with update is on an average $\frac{2}{3}$ -rd of the computation time of preconditioner without update. With average timings running into hours for very large systems, such savings are substantial.

1. INTRODUCTION

A continuous time-invariant second order linear dynamical system is of the form

$$(1.1) \quad \begin{aligned} M\ddot{x}(t) &= -D\dot{x}(t) - Kx(t) + Fu(t), \\ y(t) &= C_px(t) + C_v\dot{x}(t), \end{aligned}$$

where $M, D, K \in \mathbb{R}^{n \times n}$ are mass, damping and stiffness matrices, respectively, $F \in \mathbb{R}^{n \times m}$, $C_p, C_v \in \mathbb{R}^{q \times n}$ are constant matrices. In (1.1), $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ is the input, and $y(t) \in \mathbb{R}^q$ is the output. If m and q both are one, then we have a Single-Input Single-Output (SISO) system. Otherwise (m and $q > 1$) the system is called Multi-Input Multi-Output (MIMO). We assume the case of proportional damping, i.e., $D = \alpha M + \beta K$, where the coefficients α and β are chosen based on experimental results [4, 8]. For our derivations, the system matrices M, D and K need not hold any specific property (e.g., symmetry, positive definiteness etc.).

2010 *Mathematics Subject Classification.* Primary 34C20, 65F10, 65L20.

Key words and phrases. Model Order Reduction, Global Arnoldi Algorithm, Moment Matching, Iterative Methods, Preconditioner and Stability Analysis.

This work was supported by DAAD grant number A/14/04422 under the IIT-TU9 exchange of faculty program.

It is assumed that the order n of the system (1.1) is extremely high. The simulation of large dynamical systems can be unmanageable due to high demands on computational resources, which is the main motivation for model reduction. The goal of model reduction is to produce a low dimensional system that has, as best as possible, the same characteristics as the original system but whose simulation requires significantly less computational effort. The reduced system of (1.1) is described by

$$(1.2) \quad \begin{aligned} \hat{M}\ddot{\hat{x}}(t) &= -\hat{D}\dot{\hat{x}}(t) - \hat{K}\hat{x}(t) + \hat{F}u(t), \\ \hat{y}(t) &= \hat{C}_p\hat{x}(t) + \hat{C}_v\dot{\hat{x}}(t), \end{aligned}$$

where $\hat{M}, \hat{K}, \hat{D} \in \mathbb{R}^{r \times r}$, $\hat{F} \in \mathbb{R}^{r \times m}$, $\hat{C}_p, \hat{C}_v \in \mathbb{R}^{q \times r}$ and $r \ll n$. The damping property of the original system needs to be reflected in the reduced system. That is, $\hat{D} = \alpha\hat{M} + \beta\hat{K}$ is required, where α and β remain unchanged from the original system.

Model reduction can be done in many ways, see, e.g., [2]. We will focus on a projection based method, specifically Galerkin projection [8]. For this a matrix $V \in \mathbb{R}^{n \times r}$ with orthonormal columns is chosen and the system (1.1) is projected

$$(1.3) \quad \begin{aligned} V^T(MV\ddot{\hat{x}}(t) + DV\dot{\hat{x}}(t) + KV\hat{x}(t) - Fu(t)) &= 0, \\ \hat{y}(t) &= C_pV\hat{x}(t) + C_vV\dot{\hat{x}}(t). \end{aligned}$$

Comparing (1.3) with (1.2) yields

$$(1.4) \quad \begin{aligned} \hat{M} &= V^T M V, \quad \hat{D} = V^T D V, \quad \hat{K} = V^T K V, \quad \hat{F} = V^T F, \\ \hat{C}_p &= C_p V \text{ and } \hat{C}_v = C_v V. \end{aligned}$$

The matrix V can be obtained in many ways, see, e.g., [2]. The focus in this paper will be on the Adaptive Iterative Rational Global Arnoldi (AIRGA) algorithm [8] in which V is generated by an Arnoldi based approach.

The main contributions of this paper are as follows: Section 2 summarizes the AIRGA model reduction process which uses a direct solver for solving the linear systems arising in each iteration step. In Section 3, we discuss the use of iterative solvers and preconditioners for these linear systems. Preconditioned iterative solvers are a good choice here since they scale well. They have time complexity $\mathcal{O}(n \cdot nnz)$, where n is the size of the system and nnz is the number of nonzeros in system matrices as compared to $\mathcal{O}(n^3)$ for direct solvers [17]. The choice of iterative algorithm is problem dependent. We show that Sparse Approximate Inverse (SPAI) preconditioners are well suited for solving the linear systems arising in the model reduction process. These linear systems change at each model reduction iteration, but this change is small. Exploiting this fact we propose a cheap preconditioner update. Using an iterative solver introduces additional errors in the computation since the linear systems are not solved exactly. Hence, we discuss the stability of AIRGA in Section 4. In Section 5, an numerical experiment is given to support our preconditioned iterative solver theory. The cheap updates to the SPAI preconditioner, with CG as the underlying iterative algorithm, leads to about $\frac{1}{3}$ -rd savings in time. Finally, we give some conclusions and point out future work in Section 6.

For the rest of this paper, $\|\cdot\|_F$ denotes the Frobenius norm, $\|\cdot\|$ the 2-norm, $\|\cdot\|_{H_2}$ the H_2 -norm, and $\|\cdot\|_{H_\infty}$ the H_∞ -norm [2]. Also, qr denotes the QR factorization [21].

2. ARNOLDI BASED PROJECTION METHOD

In this section, we first describe how to obtain V such that the first few moments of the transfer functions of the original and the reduced order transfer function are matched. We then state the AIRGA algorithm [8] based on this approach.

The transfer function of (1.1) is given by

$$H(s) = (C_p + sC_v)(s^2M + sD + K)^{-1}F = (C_p + sC_v)X(s),$$

where $X(s) = (s^2M + sD + K)^{-1}F$ is the state variable in frequency domain. The power series expansion of state variable $X(s)$ around expansion point $s_0 \in \mathbb{R}$ is given as [24]

$$(2.1) \quad X(s) = \sum_{j=0}^{\infty} X^{(j)}(s_0)(s - s_0)^j,$$

where,

$$(2.2) \quad \begin{aligned} X^{(0)}(s_0) &= (s_0^2M + s_0D + K)^{-1}F, \\ X^{(1)}(s_0) &= (s_0^2M + s_0D + K)^{-1}(-(2s_0M + D))X^{(0)}(s_0), \\ X^{(2)}(s_0) &= (s_0^2M + s_0D + K)^{-1}[-(2s_0M + D)X^{(1)}(s_0) - MX^{(0)}(s_0)], \\ &\vdots \\ X^{(j)}(s_0) &= (s_0^2M + s_0D + K)^{-1}[-(2s_0M + D)X^{(j-1)}(s_0) - MX^{(j-2)}(s_0)]. \end{aligned}$$

Here, $X^{(j)}(s_0)$ is called the j^{th} -order system moment of $X(s)$ at s_0 .

Similarly, the transfer function of the reduced system (1.2) is given by

$$\hat{H}(s) = (\hat{C}_p + s\hat{C}_v)\hat{X}(s),$$

where $\hat{X}(s) = (s^2\hat{M} + s\hat{D} + \hat{K})^{-1}\hat{F}$. The power series expansion of the reduced state space $\hat{X}(s)$ around expansion point $s_0 \in \mathbb{R}$ is

$$(2.3) \quad \hat{X}(s) = \sum_{j=0}^{\infty} \hat{X}^{(j)}(s_0)(s - s_0)^j.$$

Here, $\hat{X}^{(j)}(s_0)$ is defined analogously to the $X^{(j)}(s_0)$. It is called the j^{th} -order system moment of $\hat{X}(s)$ at s_0 .

The goal of moment-matching approaches is to find a reduced order model such that the first few moments of (2.1) and (2.3) are matched, that is, $X^{(j)}(s_0) = \hat{X}^{(j)}(s_0)$ for $j = 0, 1, 2, \dots, t$ for some t .

Define

$$\begin{aligned} \mathcal{P}_1 &= -(s_0^2M + s_0D + K)^{-1}(2s_0M + D), \\ \mathcal{P}_2 &= -(s_0^2M + s_0D + K)^{-1}M, \\ \mathcal{Q} &= (s_0^2M + s_0D + K)^{-1}F, \end{aligned}$$

then from (2.2) we have

$$\begin{aligned} X^{(0)}(s_0) &= \mathcal{Q}, \\ X^{(1)}(s_0) &= \mathcal{P}_1X^{(0)}(s_0), \quad \text{and} \\ X^{(j)}(s_0) &= \mathcal{P}_1X^{(j-1)}(s_0) + \mathcal{P}_2X^{(j-2)}(s_0) \end{aligned}$$

for $j \geq 2$.

The second order Krylov subspace [3] is defined as

$$\mathbb{G}_j(\mathcal{P}_1, \mathcal{P}_2, \mathbf{Q}) = \text{span}\{\mathbf{Q}, \mathcal{P}_1\mathbf{Q}, (\mathcal{P}_1^2 + \mathcal{P}_2)\mathbf{Q}, \dots, \mathcal{S}_j(\mathcal{P}_1, \mathcal{P}_2)\mathbf{Q}\},$$

where $\mathcal{S}_j(\mathcal{P}_1, \mathcal{P}_2) = \mathcal{P}_1 \cdot \mathcal{S}_{j-1}(\mathcal{P}_1, \mathcal{P}_2) + \mathcal{P}_2 \cdot \mathcal{S}_{j-2}(\mathcal{P}_1, \mathcal{P}_2)$ for $j \geq 2$.

Let $\tilde{K} = (s_0^2 M + s_0 D + K)$. For the special case of proportionally damped second-order systems, it has been observed in [4]

$$\begin{aligned} \mathbb{G}_j(\mathcal{P}_1, \mathcal{P}_2, \mathbf{Q}) &= \mathbb{G}_j(-\tilde{K}^{-1}(2s_0 M + D), -\tilde{K}^{-1}M, \tilde{K}^{-1}F), \\ &= \mathbb{G}_j(-\tilde{K}^{-1}((2s_0 + \alpha)M + \beta K), -\tilde{K}^{-1}M, \tilde{K}^{-1}F), \\ &= \mathbb{K}_j(\mathcal{P}_1, \mathbf{Q}), \end{aligned}$$

where $\mathbb{K}_j(\mathcal{P}_1, \mathbf{Q})$ is the standard block Krylov subspace

$$\mathbb{K}_j(\mathcal{P}_1, \mathbf{Q}) = \text{span}\{\mathbf{Q}, \mathcal{P}_1\mathbf{Q}, \mathcal{P}_1^2\mathbf{Q}, \dots, \mathcal{P}_1^{j-1}\mathbf{Q}\}.$$

Thus, we need a good basis of $\mathbb{K}_j(\mathcal{P}_1, \mathbf{Q})$. This can be obtained efficiently by, e.g., the block or the global Arnoldi algorithm [17, 14, 18].

The AIRGA algorithm, as proposed in [8], is one of the latest methods based on the global Arnoldi method. It is given in Algorithm 1. In this method, moment matching is done at multiple expansion points s_i , $i = \{1, \dots, l\}$, rather than just at s_0 as earlier. This ensures a better reduced model in the entire frequency domain of interest.

The initial selection and further the computation of expansion points has been discussed in [8] and [13]. We adopt the choices described in Section 5.0.1 of [8]. The initial expansion points could be either real or imaginary, both of which have their merits. This is problem dependent and discussed in results section. After the first AIRGA iteration, the expansion points are chosen from the eigenvalues of the quadratic eigenvalue problem $\lambda^2 \hat{M} + \lambda \hat{D} + \hat{K}$ (at line 33).

The method is adaptive, i.e., it automatically chooses the number of moments to be matched at each expansion point s_i . This is controlled by the while loop at line 9. The variable j stores the number of moments matched. The upper bound on j is $\lceil r_{\max}/m \rceil$, where r_{\max} is the maximum dimension to which we want to reduce the state variable (input from the user), and m is the dimension of the input; see [8] for a detailed discussion. At exit of this while loop, $J = j$.

At line 3, no convergence implies that the H_2 norm of the difference between two consecutive reduced systems, computed at line 34, is greater than a certain tolerance. Similarly, at line 9, no convergence implies that the H_2 norm of the difference between two consecutive intermediate reduced systems, computed at line 26, is greater than a certain tolerance.

This algorithm requires solving a linear system at line 5 and 14. As the s_i change in each iteration step, the linear systems to be solved change in each iteration step. As discussed in Section 1, since solving such systems by direct methods is quite expensive, we propose to use iterative methods. As the change in the s_i will be small (at least after the first iteration step), we can develop a cheap update of the necessary preconditioner.

3. PRECONDITIONED ITERATIVE METHOD

There are two types of methods for solving linear systems of equations; a) direct methods and b) iterative methods. For large systems, direct methods are not

Algorithm 1 Adaptive Iterative Rational Global Arnoldi Algorithm [8]

```

1: Input:  $\{M, D, K, F, C_p, C_v, r_{\max}; S \text{ is the set initial expansion points } s_i, i = 1, \dots, l\}$ 
2:  $z = 1$ 
3: while no convergence do
4:   for each  $s_i \in S$  do
5:      $X^{(0)}(s_i) = (s_i^2 M + s_i D + K)^{-1} F$ 
6:     Compute  $QR = qr(X^{(0)}(s_i))$ ,  $X^{(0)}(s_i) = Q$ 
7:   end for
8:    $j = 1$ 
9:   while no convergence and  $j \leq \lceil r_{\max}/m \rceil$  do
10:    Let  $\sigma_j$  be expansion point corresponding to maximum moment error of reduced system at  $s_i$ 
11:     $V_j = X^{(j-1)}(\sigma_j) / \|X^{(j-1)}(\sigma_j)\|_F$ 
12:    for  $i = 1, \dots, l$  do
13:      if  $(s_i == \sigma_j)$  then
14:         $X^{(j)}(s_i) = -(s_i^2 M + s_i D + K)^{-1} M V_j$ 
15:      else  $X^{(j)}(s_i) = X^{(j-1)}(s_i)$ 
16:      end if
17:      for  $t = 1, 2, \dots, j$  do
18:         $\gamma_{t,j}(s_i) = \text{trace}(V_t^H \cdot X^{(j)}(s_i))$ 
19:         $X^{(j)}(s_i) = X^{(j)}(s_i) - \gamma_{t,j}(s_i) V_t$ 
20:      end for
21:    end for
22:     $W_i = X^{(j)}(s_i) / \|X^{(j)}(s_i)\|_F$  for  $i = \{1, \dots, l\}$ .
23:     $\tilde{W} = [W_1, W_2, \dots, W_l]$ .
24:    Compute  $\tilde{W}Y = qr(\tilde{W})$ ,  $W = \tilde{W}$ 
25:    Compute reduced system matrices  $\hat{M}, \hat{D}$ , and  $\hat{K}$  with  $V = W$  as in (1.4)
26:     $\hat{H}_{\text{Int}} = (\hat{C}_p + \sigma_j \hat{C}_v)(\sigma_j^2 \hat{M} + \sigma_j \hat{D} + \hat{K})^{-1} \hat{F}$ 
27:     $j = j + 1$ 
28:  end while
29:  Set  $J = j$  and pick  $\sigma_J$  corresponding to maximum moment error of reduced system at  $s_i$ 
30:   $V_J = X^{(J-1)}(\sigma_J) / \|X^{(J-1)}(\sigma_J)\|_F$  and  $\tilde{V} = [V_1, V_2, \dots, V_J]$ .
31:  Compute  $\tilde{V}Y = qr(\tilde{V})$ ,  $V = \tilde{V}$ 
32:  Compute reduced system matrices  $\hat{M}, \hat{D}$ , and  $\hat{K}$  with  $V$  as in (1.4), and take  $M = \hat{M}$ ,  $D = \hat{D}$ ,  $K = \hat{K}$ , for the next iteration.
33:  Choose new expansion points  $s_i$ 
34:   $\hat{H} = (\hat{C}_p + \sigma_J \hat{C}_v)(\sigma_J^2 \hat{M} + \sigma_J \hat{D} + \hat{K})^{-1} \hat{F}$ 
35:   $z = z + 1$ 
36: end while
37: Compute the remaining reduced system matrices  $\hat{F}, \hat{C}_p, \hat{C}_v$  with  $V$  as in (1.4)

```

preferred because they are too expensive in terms of storage and operation. On the other hand, iterative methods require less storage and operations than direct methods. For a large linear system $Ax = b$, with $A \in \mathbb{R}^{n \times n}$ and $b \in \mathbb{R}^n$, an iterative

method finds a sequence of solution vectors x_0, x_1, \dots, x_k which (hopefully) converges to the desired solution. Krylov subspace based methods are an important and popular class of iterative methods. If x_0 is the initial solution and $r_0 = b - Ax_0$ is the initial residual, then Krylov subspace methods find the approximate solution by projecting onto the Krylov subspace

$$\mathbb{K}_k(A, r_0) = \text{span}\{r_0, Ar_0, A^2r_0, \dots, A^{k-1}r_0\}.$$

There are many types of Krylov subspace algorithms [17]. Some popular ones include Conjugate Gradient (CG), Generalized Minimal Residual (GMRES), Minimum Residual (MINRES), and BiConjugate Gradient (BiCG). Block versions of these algorithms do exist. The choice of algorithm is problem dependent. In results section (Section 5), the coefficient matrices arising from the problem are symmetric positive definite (SPD). Hence, we use CG algorithm, which is ideal for such systems.

In Krylov subspace methods, the conditioning of the system is very important. “Conditioning pertains to the perturbation behavior of a mathematical problem [21]”. For example, in a well-conditioned problem, a small perturbation of the input leads to a small change in the output. This is not guaranteed for an ill-conditioned problem, where a small perturbation in the input may change the output drastically [21]. Preconditioning is a technique to well-condition an ill-conditioned problem. We discuss that next.

Preconditioning is a technique for improving the performance of iterative methods. It transforms a difficult system (ill-conditioned system) to another system with more favorable properties for iterative methods. For example, a preconditioned matrix may have eigenvalues clustered around one. This means that the preconditioned matrix is close to the identity matrix, and hence, the iterative method will converge faster. For a symmetric positive definite (SPD) system, the convergence rate of iterative methods depends on the distribution of the eigenvalues of the coefficient matrix. However, for a non-symmetric system, the convergence rate may depend on pseudo-spectra as well [20, 16].

If M is a nonsingular matrix which approximates A ; that is, $M \approx A^{-1}$, then the system

$$MAx = Mb$$

may be faster to solve than the original one. The above system represents preconditioning from left. Similarly, right and split preconditioning is given by two equations below, respectively.

$$AM\tilde{x} = b, \quad x = M\tilde{x} \quad \text{and} \quad M_1AM_2\tilde{x} = M_1b, \quad x = M_2\tilde{x}.$$

The type of preconditioning technique to be used depends on the problem properties as well as on the choice of the iterative solver. For example, for SPD systems, MA , AM , M_1AM_2 all have same eigenvalue spectrum, and hence, left, right and split preconditioners behave the same way, respectively. For a general system, this need not be true [6].

Besides making the system easier to solve by an iterative method, a preconditioner should be cheap to construct and apply. Some existing preconditioning techniques include Successive Over Relaxation, Polynomial, Incomplete Factorizations, Sparse Approximate Inverse (SPAI), and Algebraic Multi-Grid [6, 7, 9, 10].

We use SPAI preconditioner here since these (along with incomplete factorizations) are known to work in the most general setting. Also, SPAI preconditioners are easily parallelizable, hence, have an edge over incomplete factorization based preconditioners [23].

In Section 3.1 we summarize the SPAI preconditioner from [10] and we discuss the use of SPAI in the AIRGA algorithm. Since the change in the coefficient matrix of the linear system to be solved is small from one step of AIRGA to the next, we update the preconditioner from one step to the next. This aspect is covered in Section 3.2.

3.1. Sparse Approximate Inverse (SPAI) Preconditioner. In constructing a preconditioner $P_i^{(z)}$ for a coefficient matrix $\mathcal{K}_i^{(z)}$ at the z^{th} outer AIRGA iteration (i.e., $\mathcal{K}_i^{(z)} = s_i^2 M + s_i D + K$), we would like $P_i^{(z)} \mathcal{K}_i^{(z)} \approx I$ (for left preconditioning) and $\mathcal{K}_i^{(z)} P_i^{(z)} \approx I$ (for right preconditioning). SPAI preconditioners find $P_i^{(z)}$ by minimizing the associated error norm $\|I - P_i^{(z)} \mathcal{K}_i^{(z)}\|$ or $\|I - \mathcal{K}_i^{(z)} P_i^{(z)}\|$ for a given sparsity pattern. If the norm used is Frobenius norm, then the minimization function will be

$$\min_{P_i^{(z)} \in S} \|I - \mathcal{K}_i^{(z)} P_i^{(z)}\|_F,$$

where S is a set of certain sparse matrices. The above approach produces a right approximate inverse. Similarly, a left approximate inverse can be computed by solving the minimization problem $\|I - P_i^{(z)} \mathcal{K}_i^{(z)}\|_F$. For non-symmetric matrices, the distinction between left and right approximate inverses is important. There are some situations where it can be difficult to find a right approximate inverse but finding a left approximate inverse can be easy. Whether left or right preconditioning should be used is problem dependent [17]. Since the SPAI preconditioner was originally proposed for right preconditioning [10], we focus on the same here. Similar derivation can be done for the left preconditioning as well. The minimization problem can be rewritten as

$$(3.1) \quad \min \|I - \mathcal{K}_i^{(z)} P_i^{(z)}\|_F^2 = \min \sum_{j=1}^n \|e^{(j)} - \mathcal{K}_i^{(z)} p_i^{(j)}\|_2^2,$$

where $p_i^{(j)}$ and $e^{(j)}$ are j^{th} columns of the $P_i^{(z)}$ matrix and I (identity matrix), respectively. The minimization problem (3.1) is essentially just one least squares problem, to be solved for n different right-hand sides. Here it is solved iteratively.

The algorithm for computing a SPAI preconditioner for right preconditioning is given in Algorithm 2. The inputs to this algorithm is $\mathcal{K}_i^{(z)}$ (coefficient matrix) and tol (stopping residual of the minimization problem for each column); tol is picked based on experience. This is ALGORITHM 2.5 of [10] with two minor differences.

First, we do not list the code related to sparsity pattern matching (for obtaining a sparse preconditioner) because the goal here is to motivate SPAI update, and for our problems the original matrix is very sparse so the preconditioner stays sparse any ways. This aspect can be easily incorporated. Second, we use a *While* loop at line 6 of Algorithm 2 instead of a *For* loop. This is because with a *For* loop one has to decide the stopping count in advance (which is chosen heuristically). We use a more certain criteria. That is, residual of the minimization problem for each

Algorithm 2 : Sparse Approximate Inverse (SPAI) Preconditioner [10]

```

1: Input:  $\{\mathcal{K}_i^{(z)}, tol\}$ 
2:  $P_i^{(z)} = \alpha I$  where  $\alpha = \frac{\text{trace}(\mathcal{K}_i^{(z)})}{\text{trace}(\mathcal{K}_i^{(z)} (\mathcal{K}_i^{(z)})^T)}$  and  $n$  is the dimension of  $\mathcal{K}_i^{(z)}$ 
3: for  $j = 1, \dots, n$  do
4:   Define  $p_i^{(j)} = P_i^{(z)} e^{(j)}$ 
5:    $r = e^{(j)} - \mathcal{K}_i^{(z)} p_i^{(j)}$ 
6:   while  $\|r\| > tol$  do
7:      $d = P_i^{(z)} r$ 
8:      $w = \mathcal{K}_i^{(z)} d$ 
9:      $\alpha = \frac{(r, w)}{(w, w)}$ 
10:     $p_i^{(j)} = p_i^{(j)} + \alpha d$ 
11:     $r = r - \alpha w$ 
12:   end while
13: end for

```

column less than tol . This is linear cost (for each column) and we are doing such computation anyways.

The initial guess for approximate inverse $P_i^{(z)}$ is usually taken as αI where $\alpha = \text{trace}(\mathcal{K}_i^{(z)}) / \text{trace}(\mathcal{K}_i^{(z)} (\mathcal{K}_i^{(z)})^T)$ (see line 2). This initial scaling factor α is minimizes the spectral radius of $(I - \alpha \mathcal{K}_i^{(z)})$ [7, 10, 15].

The AIRGA algorithm with the SPAI preconditioner is given in Algorithm 3. Here, we only show those parts of AIRGA algorithm that require changes.

Algorithm 3 : AIRGA Algorithm with SPAI Preconditioner

```

1: while no convergence do
2:   for  $i = 1, \dots, l$  do
3:     Let  $\mathcal{K}_i^{(z)} = (s_i^2 M + s_i D + K)$ 
4:     Compute preconditioner  $P_i^{(z)}$  by solving  $\min \|I - \mathcal{K}_i^{(z)} P_i^{(z)}\|_F^2$ 
5:     Solve  $\mathcal{K}_i^{(z)} P_i^{(z)} \tilde{X}^{(0)}(s_i) = F$  with  $X^{(0)}(s_i) = P_i^{(z)} \tilde{X}^{(0)}(s_i)$ 
6:   end for
7:    $j = 1$ 
8:   while no convergence and  $j \leq \lceil r_{\max}/m \rceil$  do
9:     for  $i = 1, \dots, l$  do
10:      Only right hand sides are changing, so above preconditioner  $P_i^{(z)}$  can
      be applied as it is, i.e.,
      Solve  $\mathcal{K}_i^{(z)} P_i^{(z)} \tilde{X}^{(j)}(s_i) = MV_j$  with  $X^{(j)}(s_i) = P_i^{(z)} \tilde{X}^{(j)}(s_i)$ 
11:    end for
12:   end while
13:    $j = j+1$ 
14: end while

```

3.2. SPAI Update Preconditioner. Let $\mathcal{K}_{old} = s_{old}^2 M + s_{old} D + K$ and $\mathcal{K}_{new} = s_{new}^2 M + s_{new} D + K$ be two coefficient matrices for different expansion points s_{old} and s_{new} , respectively. These expansion points can be at the same or different AIRGA iteration. If the difference between \mathcal{K}_{old} and \mathcal{K}_{new} is small, then one can exploit this while building preconditioners for this sequence of matrices. This has been considered in the quantum Monte Carlo setting [1] and for model reduction of first order linear dynamical systems [12, 25].

Let P_{old} be a good initial preconditioner for \mathcal{K}_{old} . As will be seen, a cheap preconditioner update can be obtained by asking for $\mathcal{K}_{old} P_{old} \approx \mathcal{K}_{new} P_{new}$, where $old, new = \{1, \dots, l\}$ and, as earlier, l denotes the number of expansion points. Expressing \mathcal{K}_{new} in terms of \mathcal{K}_{old} , we get

$$\mathcal{K}_{new} = \mathcal{K}_{old}(I + (s_{new}^2 - s_{old}^2)\mathcal{K}_{old}^{-1}M + (s_{new} - s_{old})\mathcal{K}_{old}^{-1}D).$$

Now we enforce $\mathcal{K}_{old} P_{old} = \mathcal{K}_{new} P_{new}$ or

$$\begin{aligned} \mathcal{K}_{old} P_{old} &= \mathcal{K}_{old}(I + (s_{new}^2 - s_{old}^2)\mathcal{K}_{old}^{-1}M + (s_{new} - s_{old})\mathcal{K}_{old}^{-1}D) \\ &\quad \cdot (I + (s_{new}^2 - s_{old}^2)\mathcal{K}_{old}^{-1}M + (s_{new} - s_{old})\mathcal{K}_{old}^{-1}D)^{-1} P_{old} \\ &= \mathcal{K}_{new} P_{new}, \end{aligned}$$

where $P_{new} = (I + (s_{new}^2 - s_{old}^2)\mathcal{K}_{old}^{-1}M + (s_{new} - s_{old})\mathcal{K}_{old}^{-1}D)^{-1} P_{old}$.

Let $Q_{new} \approx (I + (s_{new}^2 - s_{old}^2)\mathcal{K}_{old}^{-1}M + (s_{new} - s_{old})\mathcal{K}_{old}^{-1}D)^{-1}$, then the above implies $\mathcal{K}_{old} P_{old} \approx \mathcal{K}_{new} Q_{new} P_{old}$ or $\mathcal{K}_{old} \approx \mathcal{K}_{new} Q_{new}$. This leads us to the following idea: instead of solving for P_{new} from $\mathcal{K}_{old} P_{old} = \mathcal{K}_{new} P_{new}$, we solve a simpler problem

$$\min \|\mathcal{K}_{old} - \mathcal{K}_{new} Q_{new}\|_F^2 = \min \sum_{j=1}^n \left\| k_{old}^{(j)} - \mathcal{K}_{new} q_{new}^{(j)} \right\|_2^2,$$

where $k_{old}^{(j)}$ and $q_{new}^{(j)}$ denote the j^{th} columns of \mathcal{K}_{old} and Q_{new} , respectively. Compare this minimization problem with the one in SPAI (Equation (3.1) in Section 3.1). Earlier, we were finding the preconditioner P_{new} for \mathcal{K}_{new} by solving $\min \|I - \mathcal{K}_{new} P_{new}\|_F^2$. Here, we are finding the preconditioner P_{new} (i.e., $P_{new} = Q_{new} P_{old}$) by solving $\min \|\mathcal{K}_{old} - \mathcal{K}_{new} Q_{new}\|_F^2$. The second formulation is much easier to solve since in the first \mathcal{K}_{new} could be very different from I , while in the second \mathcal{K}_{new} and \mathcal{K}_{old} are similar (change only in the expansion points).

The SPAI algorithm (Algorithm 2) adapted for finding the preconditioner by minimizing this new expression is given in Algorithm 4. The inputs to this algorithm include \mathcal{K}_{old} , \mathcal{K}_{new} , and tol (stopping residual of the minimization problem for each column); tol is picked based on experience. The initial guess for the approximate inverse Q_{new} is usually taken as αI . Similar to before, α is chosen to minimize the

Algorithm 4 : SPAI Update Preconditioner

```

1: Input:  $\{\mathcal{K}_{old}, \mathcal{K}_{new}, tol\}$ 
2:  $Q_{new} = \alpha I$ 
   where  $\alpha = \frac{1}{2} \cdot \frac{\text{trace}(\mathcal{K}_{old}^T \mathcal{K}_{new} + \mathcal{K}_{new}^T \mathcal{K}_{old})}{\text{trace}(\mathcal{K}_{new}^T \mathcal{K}_{new})}$  and  $n$  is the dimension of  $\mathcal{K}_{new}$ 
3: for  $j = 1, \dots, n$  do
4:   Define  $q^{(j)} = Q_{new} e^{(j)}$ 
5:    $r = k_{old}^{(j)} - \mathcal{K}_{new} q^{(j)}$ 
6:   while  $\|r\| > tol$  do
7:      $d = Q_{new} r$ 
8:      $w = \mathcal{K}_{new} d$ 
9:      $\alpha = \frac{(r, w)}{(w, w)}$ 
10:     $q^{(j)} = q^{(j)} + \alpha d$ 
11:     $r = r - \alpha w$ 
12:   end while
13: end for

```

spectral radius of $(\mathcal{K}_{old} - \alpha \mathcal{K}_{new})$:

$$\begin{aligned}
& \frac{\partial}{\partial \alpha} \|\mathcal{K}_{old} - \alpha \mathcal{K}_{new}\|_F^2 = 0 \quad \text{or} \\
& \frac{\partial}{\partial \alpha} \|\mathcal{K}_{old} - \alpha \mathcal{K}_{new}\|_F^2 = \frac{\partial}{\partial \alpha} \text{trace}(\mathcal{K}_{old} - \alpha \mathcal{K}_{new})^T (\mathcal{K}_{old} - \alpha \mathcal{K}_{new}) = 0 \quad \text{or} \\
& \frac{\partial}{\partial \alpha} \text{trace}[\mathcal{K}_{old}^T \mathcal{K}_{old} - \alpha \mathcal{K}_{old}^T \mathcal{K}_{new} - \alpha \mathcal{K}_{new}^T \mathcal{K}_{old} + \alpha^2 \mathcal{K}_{new}^T \mathcal{K}_{new}] = 0 \quad \text{or} \\
& 2\alpha \cdot \text{trace}(\mathcal{K}_{new}^T \mathcal{K}_{new}) = \text{trace}(\mathcal{K}_{old}^T \mathcal{K}_{new} + \mathcal{K}_{new}^T \mathcal{K}_{old}) \quad \text{or} \\
& \alpha = \frac{1}{2} \cdot \frac{\text{trace}(\mathcal{K}_{old}^T \mathcal{K}_{new} + \mathcal{K}_{new}^T \mathcal{K}_{old})}{\text{trace}(\mathcal{K}_{new}^T \mathcal{K}_{new})}.
\end{aligned}$$

In AIRGA, this update to the preconditioner can be done in two ways. To understand these ways, let us look at how expansion points change in AIRGA. Table 1 shows changing expansion points (labeled as Exp Pnt) for two iterations of the while loop at line 3 of Algorithm 1. First, we can update the preconditioner when expansion points change from $s_1^{(1)}$ to $s_2^{(1)}$, $s_2^{(1)}$ to $s_3^{(1)}$, $s_3^{(1)}$ to $s_4^{(1)}$ and so on (horizontal update). Second, we can update the preconditioner when expansion points change from $s_1^{(1)}$ to $s_1^{(2)}$, $s_2^{(1)}$ to $s_2^{(2)}$, $s_3^{(1)}$ to $s_3^{(2)}$ and so on (vertical update). Since in AIRGA, vertical change in expansion points is less (which means vertically the coefficient matrices are close), we use this strategy. Thus, we are updating

TABLE 1. Change in Expansion Points

Outer AIRGA Itn (z)	Exp Pnt 1 ($i = 1$)	Exp Pnt 2 ($i = 2$)	Exp Pnt 3 ($i = 3$)	...	Exp Pnt l ($i = l$)
1	$s_1^{(1)}$	$s_2^{(1)}$	$s_3^{(1)}$...	$s_l^{(1)}$
2	$s_1^{(2)}$	$s_2^{(2)}$	$s_3^{(2)}$...	$s_l^{(2)}$

Algorithm 5 : AIRGA with SPAI Update Preconditioner

```

1:  $z = 1$ 
2: while no convergence do
3:   for  $i = 1$  to  $l$  do
4:     if  $z == 1$  then
5:        $\mathcal{K}_i^{(1)} = (s_i^2 M + s_i D + K)$ 
6:       Compute initial  $P_i^{(1)}$  by solving  $\min \left\| I - \mathcal{K}_i^{(1)} P_i^{(1)} \right\|_F^2$ 
7:       Solve  $\mathcal{K}_i^{(1)} P_i^{(1)} \tilde{X}^{(0)}(s_i) = F$  with  $X^{(0)}(s_i) = P_i^{(1)} \tilde{X}^{(0)}(s_i)$ 
8:     else
9:        $\mathcal{K}_i^{(z)} = (s_i^2 M + s_i D + K)$ 
10:      Compute  $Q_i^{(z)}$  by solving  $\min \left\| \mathcal{K}_i^{(z-1)} - \mathcal{K}_i^{(z)} Q_i^{(z)} \right\|_F^2$ 
11:      Solve  $\mathcal{K}_i^{(z)} \left[ Q_i^{(z)} Q_i^{(z-1)} \dots Q_i^{(2)} P_i^{(1)} \right] \tilde{X}^{(0)}(s_i) = F$ 
          with  $X^{(0)}(s_i) = \left[ Q_i^{(z)} Q_i^{(z-1)} \dots Q_i^{(2)} P_i^{(1)} \right] \tilde{X}^{(0)}(s_i)$ 
12:    end if
13:  end for
14:   $j = 1$ 
15:  while no convergence and  $j \leq \lceil r_{\max}/m \rceil$  do
16:    for  $i = 1$  to  $l$  do
17:      Only right hand sides are changing, so above preconditioners can be
        applied as it is, i.e.,
        Solve  $\mathcal{K}_i^{(z)} \left[ Q_i^{(z)} Q_i^{(z-1)} \dots Q_i^{(2)} P_i^{(1)} \right] \tilde{X}^{(j)}(s_i) = M V_j$ 
        with  $X^{(j)}(s_i) = \left[ Q_i^{(z)} Q_i^{(z-1)} \dots Q_i^{(2)} P_i^{(1)} \right] \tilde{X}^{(j)}(s_i)$ 
18:    end for
19:  end while
20:   $j = j + 1$ 
21: end while
22:  $z = z + 1$ 

```

preconditioner from $\mathcal{K}_i^{(z-1)} = \left(s_i^{(z-1)} \right)^2 M + s_i^{(z-1)} D + K$ (which is \mathcal{K}_{old}) to $\mathcal{K}_i^{(z)} = \left(s_i^{(z)} \right)^2 M + s_i^{(z)} D + K$ (which is \mathcal{K}_{new}).

The AIRGA algorithm with the SPAI update preconditioner is given in Algorithm 5. Here, we only show those parts of the AIRGA algorithm that require changes. At line 10 of Algorithm 5, after solving for $Q_i^{(z)}$, ideally one would obtain the preconditioner at the z^{th} AIGRA iteration as $P_i^{(z)} = Q_i^{(z)} P_i^{(z-1)}$. Since this involves a matrix-matrix multiplication, it would be expensive and defeat the purpose of using a SPAI update. Instead, we never explicitly build $P_i^{(z)}$ except at the first AIRGA iteration where we directly obtain $P_i^{(1)}$ (see line 6 of Algorithm 5). From the second AIRGA iteration onwards, we pass the all Q_i 's (until one reaches $P_i^{(1)}$) and $P_i^{(1)}$ to the Krylov solver so that we only require matrix-vector products (see line 11 of Algorithm 5).

4. STABILITY ANALYSIS OF AIRGA

An algorithm \tilde{f} for computing the solution of a continuous problem f on a digital computer is said to be stable [21] if

$$\tilde{f}(x) = f(\tilde{x}) \text{ for some } \tilde{x} \text{ with } \frac{\|\tilde{x} - x\|}{\|x\|} = \mathcal{O}(\epsilon_{\text{machine}}),$$

where $\epsilon_{\text{machine}}$ is the machine precision. Here, we study the stability of AIRGA algorithm with respect to the errors introduced by iterative methods.

Suppose $X^{(j)}(s_i)$ at line 5 and 14 in AIRGA algorithm (Algorithm 1) are computed using a direct method of solving linear system. This gives us the matrix V at line 31 in Algorithm 1. Let f be the functional representation of the moment matching process that uses V in AIRGA (i.e., exact AIRGA). Similarly, suppose $X^{(j)}(s_i)$ at line 5 and 14 in Algorithm 1 are computed using an iterative method of solving linear systems. Since iterative methods are inexact, i.e., they solve the linear systems upto a certain tolerance, we denote the resulting matrix V as \tilde{V} . Let \tilde{f} be the functional representation of the moment matching process that uses \tilde{V} in AIRGA (i.e., inexact AIRGA). Then, we will say that AIRGA is stable with respect to iterative solvers if

$$(4.1) \quad \tilde{f}(H(s)) = f(\tilde{H}(s)) \text{ for some } \tilde{H}(s) \text{ with}$$

$$(4.2) \quad \frac{\|H(s) - \tilde{H}(s)\|_{H_2 \text{ or } H_\infty}}{\|H(s)\|_{H_2 \text{ or } H_\infty}} = \mathcal{O}(\|Z\|),$$

where $\tilde{H}(s)$ is a perturbed original full model corresponding to the error in the linear solves for computing \tilde{V} in inexact AIRGA. This perturbation is denoted by Z . Further, we denote $f(H(s)) = \hat{H}(s)$ and $\tilde{f}(H(s)) = \tilde{\hat{H}}(s)$.

In Algorithm 1, the linear systems at line 5 are computed for different expansion points as

$$(s_i^2 M + s_i D + K)X^{(0)}(s_i) = F,$$

where $s_i \in \{s_1, s_2, \dots, s_l\}$. As discussed earlier, we solve these linear systems inexactly (i.e., by an iterative method). Let the residual associated with inexact linear solves for computing $X^{(0)}(s_i)$ be η_{0i} for $i = 1, \dots, l$

$$(4.3) \quad (s_i^2 M + s_i D + K)X^{(0)}(s_i) = F + \eta_{0i}.$$

Further, in Algorithm 1 at line 11, \tilde{V}_1 is computed as

$$(4.4) \quad \tilde{V}_1 = [X^{(0)}(s_{t_0}) / \|X^{(0)}(s_{t_0})\|],$$

where s_{t_0} is the expansion point corresponding to the maximum moment error of the reduced system.

Solving the linear systems for $X^{(j)}, j = 1, \dots, J-1$ at line 14 in Algorithm 1 inexactly yields

$$(4.5) \quad (s_i^2 M + s_i D + K)X^{(j)}(s_i) = M\tilde{V}_j + \eta_{ji} \quad \text{for } i = 1, \dots, l.$$

Next, \tilde{V}_{j+1} is computed as

$$(4.6) \quad \tilde{V}_{j+1} = [X^{(j)}(s_{t_j}) / \|X^{(j)}(s_{t_j})\|],$$

where s_{t_j} is the expansion point corresponding to the maximum moment error of the reduced system.

Finally, Galerkin projection is used to generate the reduced model (obtained by inexact AIRGA)

$$(4.7) \quad \begin{aligned} \tilde{M} &= \tilde{V}^T M \tilde{V}, \quad \tilde{D} = \tilde{V}^T D \tilde{V}, \quad \tilde{K} = \tilde{V}^T K \tilde{V}, \\ \tilde{F} &= \tilde{V}^T F, \quad \tilde{C}_p = C_p \tilde{V}, \quad \text{and} \quad \tilde{C}_v = C_v \tilde{V}, \end{aligned}$$

where $\tilde{V} = [\tilde{V}_1, \tilde{V}_2, \dots, \tilde{V}_J]$. (4.7) states $\tilde{f}(H(s))$. Now we have to find a perturbed original model whose exact solution, $f(\tilde{H}(s))$, will give the reduced model as obtained by the inexact solution of the original full model, $\tilde{f}(H(s))$. That is, find $\tilde{H}(s)$ such that $\tilde{f}(H(s)) = f(\tilde{H}(s))$. This would satisfy the first condition of stability (4.1).

Assume that \tilde{H} is given by the original matrices M and D and a perturbed matrix $\tilde{K} = K + Z$. Then, for \tilde{H} we have

$$(4.8) \quad (s_i^2 M + s_i D + (K + Z))X^{(0)}(s_i) = F \text{ for } i = 1, \dots, l.$$

Further, assume that the linear systems can be solved exactly as

$$(4.9) \quad (s_i^2 M + s_i D + (K + Z))X^{(j)}(s_i) = M\tilde{V}_j \text{ for } j = 1, \dots, J-1 \text{ and } i = 1, \dots, l.$$

Again, $\tilde{V} = [\tilde{V}_1, \tilde{V}_2, \dots, \tilde{V}_J]$ where \tilde{V}_1 and \tilde{V}_{j+1} for $j = 1, \dots, J-1$ are given by (4.4) and (4.6) since $X^{(0)}(s_i)$ and $X^{(j)}(s_i)$ for $j = 1, \dots, J-1$ and $i = 1, \dots, l$ are kept same as in (4.3) and (4.5), respectively.

As earlier, applying Galerkin projection to the perturbed original system gives

$$(4.10) \quad \begin{aligned} \hat{M} &= \tilde{V}^T M \tilde{V}, \quad \hat{D} = \tilde{V}^T D \tilde{V}, \quad \hat{K} = \tilde{V}^T (K + Z) \tilde{V} = \tilde{\hat{K}} + \tilde{V}^T Z \tilde{V}, \\ \hat{F} &= \tilde{V}^T F, \quad \hat{C}_p = C_p \tilde{V}, \quad \text{and} \quad C_v = C_v \tilde{V}. \end{aligned}$$

Our goal now is to find Z such that $\hat{K} = \tilde{\hat{K}}$ (recall, that we assumed that the error can be attributed solely to K ; M and D do not change). Comparing (4.3) with (4.8), and (4.5) with (4.9), we get

$$Z X^{(0)}(s_{t_0}) = -\eta_{0t_0} \quad \text{and} \quad Z X^{(j)}(s_{t_j}) = -\eta_{jt_j} \quad \text{for } j = 1, \dots, J-1,$$

where η_{0t_0} and η_{jt_j} are residuals corresponding to the maximum moment error of the reduced system in inexact AIRGA. We can rewrite Z as

$$Z \mathbf{X} = -\eta,$$

where $Z \in \mathbb{R}^{n \times n}$, $\mathbf{X} = [X^{(0)}(s_{t_0}), \dots, X^{(J-1)}(s_{t_{(J-1)}})] \in \mathbb{R}^{n \times mJ}$, and $\eta = [\eta_{0t_0}, \dots, \eta_{(J-1)t_{(J-1)}}] \in \mathbb{R}^{n \times mJ}$. As discussed in Section 2, the upper bound for J is $\lceil r/m \rceil$, and hence, $mJ < r$. Using the fact that $r \ll n$, we have $mJ < n$. Thus, we have an under-determined system of equations.

One solution of this is

$$(4.11) \quad Z = -\eta \mathbf{X}^T (\mathbf{X} \mathbf{X}^T)^{-1}.$$

Multiplying both sides of (4.11) with \tilde{V} , we get

$$(4.12) \quad \tilde{V}^T Z \tilde{V} = -\tilde{V}^T \eta \mathbf{X}^T (\mathbf{X} \mathbf{X}^T)^{-1} \tilde{V}.$$

For Ritz-Galerkin based iterative solvers, the solution space of linear systems is orthogonal to the residuals, i.e., $\tilde{V}_1 \perp \eta_{0t_0}$, $\tilde{V}_2 \perp \eta_{1t_1}$, \dots , and $\tilde{V}_J \perp \eta_{(J-1)t_{(J-1)}}$ [23]. Hence,

$$\begin{aligned} \tilde{V}^T \eta &= \begin{bmatrix} \tilde{V}_1^T \\ \tilde{V}_2^T \\ \vdots \\ \tilde{V}_{J-1}^T \\ \tilde{V}_J^T \end{bmatrix} \begin{bmatrix} \eta_{0t_0} & \eta_{1t_1} & \cdots & \eta_{(J-1)t_{(J-1)}} \end{bmatrix} \\ &= \begin{bmatrix} 0 & \tilde{V}_1^T \eta_{1t_1} & \cdots & \tilde{V}_1^T \eta_{(J-2)t_{(J-2)}} & \tilde{V}_1^T \eta_{(J-1)t_{(J-1)}} \\ \tilde{V}_2^T \eta_{0t_0} & 0 & \cdots & \tilde{V}_2^T \eta_{(J-2)t_{(J-2)}} & \tilde{V}_2^T \eta_{(J-1)t_{(J-1)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{V}_{J-1}^T \eta_{0t_0} & \tilde{V}_{J-1}^T \eta_{1t_1} & \cdots & 0 & \tilde{V}_{J-1}^T \eta_{(J-1)t_{(J-1)}} \\ \tilde{V}_J^T \eta_{0t_0} & \tilde{V}_J^T \eta_{1t_1} & \cdots & \tilde{V}_J^T \eta_{(J-2)t_{(J-2)}} & 0 \end{bmatrix}. \end{aligned}$$

Further, $\tilde{V}^T \eta \mathbf{X}^T$

$$\begin{aligned} &= \begin{bmatrix} 0 & \tilde{V}_1^T \eta_{1t_1} & \cdots & \tilde{V}_1^T \eta_{(J-2)t_{(J-2)}} & \tilde{V}_1^T \eta_{(J-1)t_{(J-1)}} \\ \tilde{V}_2^T \eta_{0t_0} & 0 & \cdots & \tilde{V}_2^T \eta_{(J-2)t_{(J-2)}} & \tilde{V}_2^T \eta_{(J-1)t_{(J-1)}} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \tilde{V}_{J-1}^T \eta_{0t_0} & \tilde{V}_{J-1}^T \eta_{1t_1} & \cdots & 0 & \tilde{V}_{J-1}^T \eta_{(J-1)t_{(J-1)}} \\ \tilde{V}_J^T \eta_{0t_0} & \tilde{V}_J^T \eta_{1t_1} & \cdots & \tilde{V}_J^T \eta_{(J-2)t_{(J-2)}} & 0 \end{bmatrix} \begin{bmatrix} X^{(0)}(s_{t_0})^T \\ X^{(1)}(s_{t_1})^T \\ \vdots \\ X^{(J-2)}(s_{t_{J-2}})^T \\ X^{(J-1)}(s_{t_{J-1}})^T \end{bmatrix} \\ &= \begin{bmatrix} 0 \cdot X^{(0)}(s_{t_0})^T + \tilde{V}_1^T \eta_{1t_1} X^{(1)}(s_{t_1})^T + \cdots + \tilde{V}_1^T \eta_{(J-1)t_{(J-1)}} X^{(J-1)}(s_{t_{(J-1)}})^T \\ \tilde{V}_2^T \eta_{0t_0} X^{(0)}(s_{t_0})^T + 0 \cdot X^{(1)}(s_{t_1})^T + \cdots + \tilde{V}_2^T \eta_{(J-1)t_{(J-1)}} X^{(J-1)}(s_{t_{(J-1)}})^T \\ \vdots \\ \tilde{V}_J^T \eta_{0t_0} X^{(0)}(s_{t_0})^T + \cdots + \tilde{V}_J^T \eta_{(J-2)t_{(J-2)}} X^{(J-2)}(s_{t_{(J-2)}})^T + 0 \cdot X^{(J-1)}(s_{t_{(J-1)}})^T \end{bmatrix} \end{aligned}$$

Since $\tilde{V}_1 \perp \eta_{0t_0}$, $\tilde{V}_2 \perp \eta_{1t_1}$, \dots , $\tilde{V}_J \perp \eta_{(J-1)t_{(J-1)}}$, and, due to (4.6), \tilde{V}_{j+1} is just the normalized $X^{(j)}(s_{t_j})$, we have $X^{(0)}(s_{t_0}) \perp \eta_{0t_0}$, $X^{(1)}(s_{t_1}) \perp \eta_{1t_1}$, \dots , and $X^{(J-1)}(s_{t_{(J-1)}}) \perp \eta_{(J-1)t_{(J-1)}}$. Therefore from (4.12), we get $\tilde{V}^T Z \tilde{V} = 0$. Thus, $\hat{K} = \tilde{\hat{K}}$ or

$$\tilde{f}(H(s)) = f(\tilde{H}(s)) = \tilde{\hat{H}}(s),$$

where $H(s) = (C_p + sC_v)(s^2M + sD + K)^{-1}F$, $\tilde{H}(s) = (C_p + sC_v)(s^2M + sD + (K + Z))^{-1}F$, and $\tilde{\hat{H}}(s) = (\tilde{\hat{C}}_p + s\tilde{\hat{C}}_v)(s^2\tilde{\hat{M}} + s\tilde{\hat{D}} + \tilde{\hat{K}})^{-1}\tilde{\hat{F}} = (\hat{C}_p + s\hat{C}_v)(s^2\hat{M} + s\hat{D} + \hat{K})^{-1}\hat{F}$. Thus, we have satisfied the first condition of stability.

According to the second condition of stability, given in (4.2), the difference between the original full model and the perturbed full model should be of the order of the perturbation [21]. This can be easily shown (Theorem 4.3 from [5]).

Theorem 4.1. *If $\|Z\| < \frac{1}{\|\mathcal{K}(s)^{-1}\|_{H_\infty}}$ then*

$$\|H(s) - \tilde{H}(s)\|_{H_2} \leq \frac{\|C(s)\mathcal{K}(s)^{-1}\|_{H_2} \|\mathcal{K}(s)^{-1}F\|_{H_\infty}}{1 - \|\mathcal{K}(s)^{-1}\|_{H_\infty} \|Z\|} \|Z\|,$$

where $\mathcal{K}(s) = (s^2M + sD + K)$ and $C(s) = (C_p + sC_v)$.

Hence,

$$\|H(s) - \tilde{H}(s)\|_{H_2} = \mathcal{O}(\|Z\|).$$

The above result holds in a relative sense too. This proves the stability of AIRGA. The next theorem summarizes this.

Theorem 4.2. *If the linear systems arising in AIRGA are solved by a Ritz-Galerkin based solver (i.e., the residual is orthogonal to the generated Krylov subspace) and*

$$\|(s^2M + sD + K)^{-1}\|_{H_\infty} \cdot \|Z\| < 1,$$

where Z given by (4.11), then AIRGA is stable.

5. NUMERICAL RESULTS

Consider a one dimensional beam model [4], which is of the form (1)

$$\begin{aligned} M\ddot{x}(t) + D\dot{x}(t) + Kx(t) &= Fu(t), \\ y(t) &= C_p x(t), \end{aligned}$$

where $m = q = 1$, $F \in \mathbb{R}^{n \times 1}$ and $C_p \in \mathbb{R}^{1 \times n}$. The model has proportional damping, i.e., $D = \alpha M + \beta K$, where the damping coefficients α and β belong to $(0, 1)$ [4]. We consider the model with two different sizes, $n = 2000$ and $n = 10000$.

We compute a reduced order model by the AIRGA algorithm given in Algorithm 1. We implement AIRGA in MATLAB (2014a). We take r_{\max} , i.e., the maximum dimension to which we want to reduce the system, as 30 for model size 2000 and 150 for model size 10000 based on [4]. We take three expansion points that are linearly spaced between 1 and 100 based on initial data used in [4]. As discussed in Section 3, we use iterative methods to solve the linear systems at lines 5 and 14 of Algorithm 1 instead of a direct method. Since the direct method (LU factorization) runs out of memory for large problems (> 50000) even on a high configuration server (64 GB RAM), we did not pursue it further for comparison.

For the model under consideration, M , D and K matrices are symmetric positive definite. Hence, with the initial expansion points all taken as real and positive, the coefficient matrices of the linear systems to solved $s_i^2M + s_iD + K$ are also symmetric positive definite initially. As discussed in Section 2, after the first AIRGA iteration, the expansion points are chosen from the eigenvalues of the quadratic eigenvalue problem $\lambda^2\hat{M} + \lambda\hat{D} + \hat{K}$. For our example, after the first AIRGA iteration, the eigenvalues of this quadratic eigenvalue problem turn out to be complex (see Table 1.1 in [19] that describes why this is supposed to happen even when all matrices are symmetric positive definite). Thus, we get complex expansion points.

Real, imaginary, or complex expansion points, each have their own merits (see Chapter 6 of [13]). After the first AIRGA iteration, we use real parts of the complex eigenvalues as the expansion points. This is because of three reasons. First, real expansion points give good approximation for general frequency response [13]. Second, using real or complex expansion points has no effect on the execution of the AIRGA algorithm as well as the accuracy of the reduced system. Third, real expansion points are computationally easier to implement (the difference between $\mathcal{K}_i^{(z-1)}$ and $\mathcal{K}_i^{(z)}$ is more easily quantifiable; see Section 3.2).

For ensuring stable iterative solves in AIRGA, from Theorem 4.2 we know that we need to use a Ritz-Galerkin based solver. Conjugate Gradient (CG) is the most

TABLE 2. CG iterations and computation time for model size 2000

AIRGA Iteration#	CG using SPAI		CG using SPAI Update	
	Iter	Time (secs)	Iter	Time (secs)
1	4	0.07	4	0.07
	4	0.07	4	0.07
	4	0.07	4	0.07
2	3	0.06	3	0.06
	3	0.06	3	0.06
	4	0.07	4	0.07
3	4	0.07	4	0.11
	4	0.07	4	0.11
	4	0.07	4	0.11
4	4	0.07	4	0.18
	4	0.07	4	0.18
	4	0.07	4	0.18

popular solver based on this theory. Moreover, since CG is ideal for SPD linear systems and we obtain such linear systems here, we use CG as the underlying iterative solver. Since the unpreconditioned CG required twice as many iterations as the preconditioned CG (for almost all problem sizes), and we did not pursue it further.

As discussed in Section 3, preconditioning has to be employed when iterative methods fail or have very slow convergence. We use SPAI and SPAI update as discussed in Section 3.1 and 3.2, respectively¹. That is, we use Algorithm 3 with Algorithm 2 and Algorithm 5 with Algorithm 4. The input to Algorithm 2 and Algorithm 4 is tol (besides the coefficient matrices), which we take as 0.01 based upon experience.

In Algorithms 3 and 5, at lines 1 and 2, respectively, the overall iteration (while-loop) terminates when the change in the reduced model (computed as H_2 -error between the reduced models at two consecutive AIRGA iterations) is less than a certain tolerance. We take this tolerance to be 10^{-06} based on values in [8]. There is one more stopping criteria in these algorithms, at lines 8 and 15, respectively. This checks the H_2 -error between two temporary reduced models. We take this tolerance to be 10^{-06} based on values in [8]. Since this is an adaptive algorithm, the optimal size of the reduced model is determined by the algorithm itself, and is denoted by r .

As discussed in Section 3.2, SPAI update in AIRGA is done vertically (see Algorithm 5 and Table 1). Ideally, this update should be done from AIRGA iteration 2 (since at iteration 1, there is no preceding set of expansion points). However, for

¹For SPD linear systems, incomplete Cholesky factorization based preconditioners are quite popular. However, as discussed in Section 3, these preconditioners are not easily parallelizable. Since for larger model sizes (> 10000 ; e.g., 100,000 and so on) parallelization would be needed and incomplete Cholesky preconditioners will not be able to compete with SPAI then, we do not use them here. Moreover, SPAI preconditioners are popular even for SPD systems (see [10, 11]).

TABLE 3. CG iterations and computation time for model size 10000

AIRGA Iteration#	CG using SPAI		CG using SPAI update	
	Iter	Time (secs)	Iter	Time (secs)
1	3	1.4	3	1.4
	4	1.4	4	1.5
	4	1.5	4	1.5
2	4	1.5	4	1.5
	3	1.4	3	1.4
	3	1.4	3	1.4
3	4	1.5	4	2.4
	3	1.4	3	2.3
	4	1.5	4	2.4
4	4	1.4	4	4.1
	3	1.4	3	3.7
	4	1.4	4	4.1

TABLE 4. SPAI and SPAI update computation time for model size 2000

AIRGA Iteration#	SPAI (secs)	SPAI with update (secs)
1	10.0	10.0
	35.0	35.0
	36.0	36.0
2	10.0	10.0
	11.0	11.0
	10.0	10.0
3	10.0	0.8
	11.0	6.0
	11.0	1.1
4	11.0	0.6
	10.0	4.0
	11.0	0.8

this problem change in expansion points from iteration 1 to 2 is fairly large (implying the corresponding coefficient matrices are far). Hence, we implement update from AIRGA iteration 3 onwards.

Table 2 gives the CG iteration count and time when using basic SPAI (that is without SPAI update) and SPAI with update for model size 2000. Table 3 gives the same data for model size 10000. From Tables 2 and 3 it is observed that the CG computation time for both variants of preconditioners is almost same. The computation time for CG using SPAI update is slightly higher than CG using basic SPAI from AIRGA iteration 3 onwards (for both model sizes). This is because from AIRGA iteration 3 onwards, we apply SPAI update and hence, the number of matrix-vector products increase (see lines 11 and 17 of Algorithm 5). However,

TABLE 5. SPAI and SPAI update computation time for model size 10000

AIRGA Iteration#	SPAI (secs)	SPAI with Update (secs)
1	282.0	282.0
	476.0	476.0
	530.0	530.0
2	177.0	177.0
	276.0	276.0
	170.0	170.0
3	172.0	28.0
	272.0	146.0
	280.0	37.0
4	179.0	27.0
	300.0	65.0
	179.0	22.0

TABLE 6. SPAI and SPAI update analysis for model size 2000

(A)

AIRGA Iteration (z)	Expansion Points 1		
	Value	$\ I - \mathcal{K}_1^{(z)}\ _F$	$\ \mathcal{K}_1^{(z-1)} - \mathcal{K}_1^{(z)}\ _F$
3	0.2681	110.5137	0.1124
4	0.2682	110.5170	0.0046

(B)

AIRGA Iteration (z)	Expansion Point 2		
	Value	$\ I - \mathcal{K}_2^{(z)}\ _F$	$\ \mathcal{K}_2^{(z-1)} - \mathcal{K}_2^{(z)}\ _F$
3	1.9948	701.4626	40.2943
4	1.8491	631.3391	9.6520

(C)

AIRGA Iteration (z)	Expansion Point 3		
	Value	$\ I - \mathcal{K}_3^{(z)}\ _F$	$\ \mathcal{K}_3^{(z-1)} - \mathcal{K}_3^{(z)}\ _F$
3	0.2700	110.6989	0.0398
4	0.2699	110.7225	0.0068

this slight increase in the time for SPAI with update fades when one takes the preconditioner computation time into account (see the discussion below).

Table 4 gives the time for computing the basic SPAI preconditioner and SPAI with update preconditioner for model size 2000. Table 5 gives the same data for model size 10000. From Tables 4 and 5 it is observed that considerable amount of time is saved by using SPAI with updates. As discussed in earlier paragraphs,

TABLE 7. SPAI and SPAI update analysis for model size 10000

(A)

AIRGA Iteration (z)	Expansion Points 1		
	Value	$\ I - \mathcal{K}_1^{(z)}\ _F$	$\ \mathcal{K}_1^{(z-1)} - \mathcal{K}_1^{(z)}\ _F$
3	0.2681	247.18	0.4279
4	0.2682	247.17	0.0066

(B)

AIRGA Iteration (z)	Expansion Point 2		
	Value	$\ I - \mathcal{K}_2^{(z)}\ _F$	$\ \mathcal{K}_2^{(z-1)} - \mathcal{K}_2^{(z)}\ _F$
3	1.5037	$1.0e + 03$	416.5
4	1.9415	$1.5e + 03$	116.8

(C)

AIRGA Iteration (z)	Expansion Point 3		
	Value	$\ I - \mathcal{K}_3^{(z)}\ _F$	$\ \mathcal{K}_3^{(z-1)} - \mathcal{K}_3^{(z)}\ _F$
3	0.2698	247.6	3.9324
4	0.2696	247.5	0.0602

TABLE 8. Total computation time of CG with preconditioners

Size	CG and SPAI (Min)	CG and SPAI update (Min)
2000	3.1	2.1
10000	61.2	40.7

TABLE 9. Accuracy of reduced system

Problem	n	Method	Error	r
1-D Beam Model	2000	LU	$1.2e - 06$	27
		CG with SPAI	$3.0e - 06$	27
		CG with SPAI update	$2.2e - 06$	27
	10000	LU	$1.1e - 06$	128
		CG with SPAI	$3.6e - 06$	128
		CG with SPAI update	$2.7e - 06$	128

since SPAI update is being done only from AIRGA iteration 3, saving in time is observed from this step onwards only.

We also analyze why SPAI update takes less time. As discussed in Section 3.2, SPAI update is useful when $\|I - \mathcal{K}_i^{(z)}\|_F$ is large and $\|\mathcal{K}_i^{(z-1)} - \mathcal{K}_i^{(z)}\|_F$ is small. This data for three expansion points for model size 2000 is given in Tables 6(A), 6(B) and 6(C). Similar data for model size 10000 is given in Tables 7(A), 7(B) and

7(C). For model size 2000, in Table 4 we see that at AIRGA iterations 3 and 4 computation time for SPAI update is almost one fourth of the SPAI time. This is because $\left\|\mathcal{K}_i^{(z-1)} - \mathcal{K}_i^{(z)}\right\|_F$ is very small as compared to $\left\|I - \mathcal{K}_i^{(z)}\right\|_F$ (see Tables 6(A), 6(B) and 6(C)). Similar pattern is observed for model size 10000 (see Table 5 and Tables 7(A), 7(B) and 7(C)).

Table 8 shows the total computation time for iterative solves (i.e., CG time plus the preconditioner time) when using basic SPAI preconditioner and when using SPAI with update preconditioner for model sizes 2000 and 10000. We can notice from this table that computation time of iterative solves with SPAI update is on an average $\frac{2}{3}$ -rd of the computation time of iterative solves with SPAI. This saving is larger for model size 10000 (we go from around 1 hour to 40 Minutes). Hence, larger the problem more the saving.

Table 9 lists the relative error between the original model and the reduced model as well as the size to which the model is reduced (r) when using LU factorization (direct method), CG with SPAI, and CG with SPAI updates. Since the error and r values for all the above three cases are almost the same, we can conclude that by using iterative solves in AIRGA, the quality of reduced system is not compromised.

6. CONCLUSION AND FUTURE WORK

We discussed the application of preconditioned iterative methods for solving large linear systems arising in AIRGA. The SPAI preconditioner works well here and SPAI update (where we reuse the preconditioner) leads to substantial savings. This is demonstrated by experiments on two different sizes of one dimensional beam model. We also presented conditions under which the AIRGA algorithm is stable with respect to the errors introduced by iterative methods.

Future work includes applying preconditioned iterative methods in other model reduction algorithms for second order dynamical systems (besides AIRGA). For example, Alternate Direction Implicit (ADI) methods for model reduction of second order linear dynamical systems [22]. Based upon our studies on AIRGA and ADI based methods, we also plan to propose a class of preconditioners that would work for most model reduction algorithms for second order linear dynamical systems.

ACKNOWLEDGEMENT

We would like to thank Prof. Eric de Sturler (at Department of Mathematics, Virginia Tech, Blacksburg, VA, USA) for stimulated discussion regarding stability of AIRGA.

REFERENCES

1. K. Ahuja, B. K. Clark, E. de Sturler, D. M. Ceperley, and J. Kim, *Improved scaling for quantum Monte Carlo on insulators*, SIAM Journal on Scientific Computing, **33** (2011), no. 4, 1837–1859.
2. A. Antoulas, *Approximation of large-scale dynamical systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005.
3. Z. Bai and Y. Su, *Dimension reduction of large-scale second-order dynamical systems via a second-order Arnoldi method*, SIAM Journal on Scientific Computing, **26** (2005), no. 5, 1692–1709.
4. C. Beattie and S. Gugercin, *Krylov-based model reduction of second-order systems with proportional damping*, Proceedings of the 44th IEEE Conference on Decision and Control, 2005, pp. 2278–2283.

5. C. Beattie, S. Gugercin, and S. A. Wyatt, *Inexact solves in interpolatory model reduction*, Elsevier Journal of Linear Algebra and its Applications, **436** (2012), no. 8, 2916–2943.
6. M. Benzi, *Preconditioning techniques for large linear systems: A survey*, Elsevier Journal of Computational Physics, **182** (2002), no. 2, 418 – 477.
7. M. Benzi and M. Tuma, *A comparative study of sparse approximate inverse preconditioners*, Applied Numerical Mathematics, **30** (1999), no. 2, 305 – 340.
8. T. Bonin, H. Fassbender, A. Soppa, and M. Zaeh, *A fully adaptive rational global Arnoldi method for the model-order reduction of second-order MIMO systems with proportional damping*, Elsevier Mathematics and Computers in Simulation, **122** (2016), 1–19.
9. E. Chow and Y. Saad, *Approximate inverse techniques for block-partitioned matrices*, SIAM Journal on Scientific Computing, **18** (1997), no. 6, 1657–1675.
10. ———, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM Journal on Scientific Computing, **19** (1998), no. 3, 995–1023.
11. T. George, A. Gupta, and V. Sarin, *An empirical analysis of the performance of preconditioners for SPD systems*, ACM Transactions on Mathematical Software, **38** (2012), no. 4, 24:1–24:30.
12. A. K. Grim-McNally, *Reusing and updating preconditioners for sequences of matrices*, Master’s thesis, Virginia Tech, USA, 2015.
13. E. J. Grimme, *Krylov projection methods for model reduction*, Ph.D. thesis, University of Illinois at Urbana-Champaign, Urbana, IL, USA, 1997.
14. K. Jbilou, A. Messaoudi, and H. Sadok, *Global FOM and GMRES algorithms for matrix equations*, Applied Numerical Mathematics **31** (1999), no. 1, 49 – 63.
15. C. D. Meyer, *Matrix analysis and applied linear algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.
16. N. M. Nachtigal, S. C. Reddy, and L. N. Trefethen, *How fast are nonsymmetric matrix iterations?*, SIAM Journal on Matrix Analysis and Applications, **13** (1992), no. 3, 778–795.
17. Y. Saad, *Iterative methods for sparse linear systems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2003.
18. M. Sadkane, *Block-Arnoldi and Davidson methods for unsymmetric large eigenvalue problems*, Numerische Mathematik **64** (1993), no. 1, 195–211.
19. F. Tisseur and K. Meerbergen, *The quadratic Eigenvalue problem*, SIAM Review **43** (2001), no. 2, 235–286.
20. L. N. Trefethen, *Pseudospectra of matrices*, Numerical Analysis, **91** (1991), 234–266.
21. L. N. Trefethen and D. Bau, *Numerical linear algebra*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1997.
22. M. M. Uddin, J. Saak, B. Kranz, and P. Benner, *Computation of a compact state space model for an adaptive spindle head configuration with piezo actuators using balanced truncation*, Springer Production Engineering, **6** (2012), no. 6, 577–586.
23. H. A. Van der Vorst, *Iterative krylov methods for large linear systems*, Cambridge University Press, New York, USA, 2003.
24. J. M. Wang, C. C. Chu, Q. Yu, and E. S. Kuh, *On projection-based algorithms for model-order reduction of interconnects*, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications **49** (2002), no. 11, 1563–1585.
25. S. A. Wyatt, *Issues in interpolatory model reduction: Inexact solves, second-order systems and DAEs*, Ph.D. thesis, Virginia Tech, USA, 2012.

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING, INDIAN INSTITUTE OF TECHNOLOGY
INDORE, INDIA

E-mail address: `phd1301201002@iiti.ac.in`

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING, INDIAN INSTITUTE OF TECHNOLOGY
INDORE, INDIA

E-mail address: `kahuja@iiti.ac.in`

INSTITUT COMPUTATIONAL MATHEMATICS, TECHNISCHE UNIVERSITÄT BRAUNSCHWEIG, GER-
MANY

E-mail address: `h.fassbender@tu-braunschweig.de`